②

Naval Research Laboratory

Washington, DC 20375-5320

NRL/FR/8150--94-9714

NAVSPACECOM Space Surveillance Sensor System Digital Signal Processing Receiver

Volume 3—Operating System Functions

CAROLYN F. BRYANT
TAMARA A. MYERS
CARL J. MORRIS

Space Applications Branch
Space Systems Development Department

April 6, 1994

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave Blank)*	2. REPORT DATE April 6, 1994	3. REPORT TYPE AND DATES COVERED Final Oct. 1990 - Sept. 1993

4. TITLE AND SUBTITLE

NAVSPACECOM Space Surveillance Sensor System Digital Signal Processing Receiver
Volume 3—Operating System Functions

5. FUNDING NUMBERS

PE - 12427N
TA - X-0125
WU - DN780-065

6. AUTHOR(S)

Carolyn F. Bryant, Tamara A. Myers, and Carl J. Morris

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Research Laboratory
Washington, DC 20375-5320

8. PERFORMING ORGANIZATION REPORT NUMBER

NRL/FR/8150--94-9714

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Naval Space Command
Dahlgren, VA

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This is a system description of the Naval Space Command (NAVSPACECOM) Space Surveillance Sensor System Digital Signal Processing Receiver (DSPR). Formerly known as NAVSPASUR, the Space Surveillance system began as an advanced research project in June 1958, was commissioned as an operational Naval command in February 1961, and is operated by NAVSPACECOM's Space Surveillance Processing Center in Dahlgren, Virginia. The DSPR is a real-time radar data acquisition and analysis system. Its function is to detect, with no prior information, all space objects whose orbits cross the continental United States and to compute their subsequent orbits. It provides vital satellite information in support of national defense mission objectives of space intelligence, satellite attack warning, satellite intercept support, and space mission support. This system description was prepared as part of a modernization program that has replaced DSPR hardware for which parts are no longer available.

Volume 3 describes the operating system functions required by the applications software. Volume 4 describes the hardware interfaces between the major subsystems of the DSPR and identifies critical timing paths and interrupts between subsystems. Previously published, Volume 1 (NRL/FR/8154--93-9577) presents an overview of the hardware and software of the DSPR system, and Volume 2 (NRL/FR/8154--93-9578) discusses the function and capabilities of individual software and hardware components of each subsystem.

14. SUBJECT TERMS

Radar
Satellite
Interferometry

15. NUMBER OF PAGES

20

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

CONTENTS

NAVSPACECOM SPACE SURVEILLANCE SENSOR SYSTEM DIGITAL SIGNAL PROCESSING RECEIVER

Volume 3—Operating System Functions

1. INTRODUCTION

This is Volume 3 of a four-volume system description of the Naval Space Command (NAVSPACECOM) Space Surveillance Sensor System Digital Signal Processing Receiver (DSPR) hardware and software. The hardware was developed by the Naval Research Laboratory (NRL) for the NAVSPACECOM Space Surveillance System (formerly NAVSPASUR). The software was designed by NRL and developed jointly by Digital Equipment Corporation (DEC) and NRL. The modernized software described in this volume was designed and developed by NRL.

The NAVSPACECOM Space Surveillance system began as an advanced research project in June 1958. In October 1960, the project was transferred from the Advanced Research Projects Agency (ARPA) to the Navy, and was subsequently commissioned as an operational Naval command in February 1961. Since then, it has been operated by the NAVSPACECOM Space Surveillance Processing Center in Dahlgren, Virginia. The Processing Center is responsible to the Chief of Naval Operations for support to the operating forces of the United States Navy, and is under the operational control of the U.S. Space Command, Colorado Springs, Colorado, for those space object data collection functions that are part of the National Space Detection and Tracking System (SPADATS).

1.1 Purpose and Scope

This volume describes the operating system functions that are required by the applications software of the DSPR system. The operating system runs in the DEC VAX 4000 Model 200 (VAX 4200) minicomputers that serve as the central processors for the DSPR system. Volume 1 presents a hardware and software overview of the DSPR system. Volume 2 describes the function and capabilities of individual software and hardware components of the system. Volume 4 discusses hardware interfaces between the major subsystems of the DSPR.

The documentation supports a project that has replaced DSPR hardware for which parts are no longer available. Hardware replaced include the DEC PDP-11/60 minicomputers that served as the central processors for the DSPR system, the Floating Point Systems Array Processors that performed fast Fourier transforms on data from the alert antennas, and various interface hardware. New hardware installed includes the VAX 4200 minicomputers, CSPI MAP 4000 array processors, and updated interface hardware. All new hardware and associated software were required to interface with the rest of the existing system. The new software was required to replicate all the functions of the existing software.

Manuscript approved January 31, 1994.

The set of documents describes the DSPR system in terms of its current functionality with modernized hardware and software. It will serve as a baseline description from which future specifications for upgrades to hardware and software may be drawn.

1.2 System Overview

The Digital Signal Processing Receiver is a real-time radar data acquisition and analysis system. The function of the NAVSPACECOM Space Surveillance system is to detect, with no prior information, all space objects whose orbits cross the continental United States and to compute their subsequent orbits.

The Space Surveillance system is a multistatic continuous-wave radar system operating as a large radio interferometer, with nine stations located along a great-circle path across the southern United States. The inclination of the great circle is 33.57° with respect to the equator.

The system consists of three transmitters and six receivers. The stations are located as follows:

Transmitters:	Jordan Lake Station, Wetumpka, Alabama
	Lake Kickapoo Station, Archer City, Texas
	Gila River Station, Maricopa, Arizona
Receivers:	Tattnall Station, Glennville, Georgia
	Hawkinsville Station, Hawkinsville, Georgia (high altitude)
	Silver Lake Station, Hollandale, Mississippi
	Red River Station, Lewisville, Arkansas
	Elephant Butte Station, Truth or Consequences, New Mexico (high altitude)
	San Diego Station, Chula Vista, California

Each transmitting station radiates a continuous wave of radio energy that combines with the other transmitting stations' beams to form the Space Surveillance "fence." When an object, such as a satellite, enters the fence, a small fraction of the radio energy is reflected to one or more of the receiver sites. The receiving stations use large multiple-array interferometers to detect the reflected signal and to measure its angle of arrival. The transmitter and receiver arrays are cross-polarized to prevent the transmitted energy from reaching the receivers without having been reflected from a space object. Each receiving station transmits phase and amplitude data, along with frequency identifiers, statistical measures, and time stamps to the Space Surveillance Processing Center via a dedicated telephone line, where local direction angles for each object are computed.

The antenna data available at the receiver stations can be processed to produce three types of data: full-Doppler, half-Doppler, and quarter-Doppler. These three types are also referred to as low-altitude, mid-altitude, and high-altitude, respectively. The output of the full-Doppler data is far more important than either the half- or quarter-Doppler data. For this reason, production of the full-Doppler data by the DSPR system takes priority over half- and quarter-Doppler processing. The purpose, then, of each DSPR is to provide a continuous stream of full-Doppler data to the Space Surveillance Processing Center at Dahlgren, Virginia.

1.3 System Philosophy

The DSPR system philosophy stems directly from the system purpose. This system is designed to run continuously, with any single component failure resulting in, at most, a very short (less than a minute) interruption in full-Doppler data processing. For this reason, each DSPR actually consists of two systems that duplicate each other. The primary system normally handles the full-Doppler data from the antennas. The secondary system normally handles the half- and quarter-Doppler data. If the primary system malfunctions and is no longer able to process data, the secondary system detects the failure and declares a "primary system failure." The secondary system then reinitializes itself to become a primary system. This reinitialization process occurs only in the secondary system. If, under normal conditions, the secondary system malfunctions, the primary system detects the failure, but does not attempt to process the half- and quarter-Doppler data. This dual system philosophy is embedded in every aspect of the DSPR system.

1.4 DSPR Subsystems

For the purposes of this document, the DSPR is treated as consisting of the following subsystems:

System Monitor and Control is responsible for running the software that monitors and coordinates all the actions of the DSPR. This includes initializing and controlling all the other subsystems described below and serving as the interface between the operators and the system.

Target Detection and Selection performs the initial detection of targets that enter the Space Surveillance fence. It analyzes each target's reflected signal and compiles the data into target lists that identify the signals by frequency and relative signal strength. These lists are passed to the central controller, which then initiates the interferometer data collection tasks.

Interferometer Data Collection comes into operation once a list of targets is generated by the Target Detection and Selection subsystem. Then, up to three targets are selected by each central controller, and the digital filters for that system are tuned to gather data. The accumulated data are provided to the central controller for formatting and transmission to the Space Surveillance Processing Center.

Data Processing converts the interferometer antenna data to phase and amplitude measurements and formats them for transmission to the Space Surveillance Processing Center.

Data Line Communications is responsible for interfacing the DSPR system to the communications line to the Space Surveillance Processing Center. Its function is to transfer data between the DSPR and the Processing Center with the necessary communications protocol. Communications are managed by a DSV11 synchronous communications controller, and data are transferred by a Codex V.3225 modem through a dedicated line at a rate of 9600 bits per second.

Interprocessor Communications and Failure Detection provides communication between the central controller of the primary system and that of the secondary system. The two central controllers communicate with each other through an Ethernet line. This subsystem is used by each half of the DSPR to detect failures in the other half and to coordinate operations between the two systems.

System Timing maintains the DSPR system timing accuracy and provides time stamping for data that are sent to the Space Surveillance Processing Center. The system time has both a hardware and software component. The software clock is initialized by reading a Hewlett-Packard 59309A HP-IB digital clock (to obtain the month, day, hour, minute, and second) and the VAX 4200's internal clock (to obtain the year), and is updated once per second by the time stamp procedure, which controls a KWV11C programmable real-time clock. A Hewlett-Packard 5061B cesium beam frequency standard provides precise 5-MHz and 1-pulse-per-second (pps) signals.

Operational Tests are on-line confidence tests used to exercise one or more components of the DSPR system. They are selected and run by the system operators, or may be initiated automatically during a system boot or reinitialization.

Utility Bus Control directs and supervises activity on the utility bus. This subsystem is used to configure the customized hardware of the DSPR system. These items include the timing generator, local oscillators, data distribution and test card, array processor input card, and several command and status modules that are connected to the utility bus.

2. OPERATING SYSTEM OVERVIEW

The operating system used in the VAX 4200 central processing unit (CPU) is VMS (virtual memory system) version 5.4-3. VMS is an interactive operating system that includes full support for the DRV1W general purpose digital input/output interfaces. User communication with VMS is carried out through the Digital Command Language (DCL).

VMS is designed as a general purpose, multiuser, multiprocessing operating system. It is based on DEC's Virtual Address Extension (VAX) concept that allows over four billion bytes of virtual address space to be available to the programmer, although the physical main memory of the CPU is not nearly as large. A memory management scheme handles details of program storage, switching into and out of main memory the context, or environment, of the various programs or processes that are running in the CPU at any one time.

During program development, the VAX Fortran Version 5 optimizing compiler, with its associated run-time support routines, and the CSPI MAP 4000 MBFORT Fortran compiler and its scientific subroutine library are used in conjunction with VMS.

2.1 Functions of the Central Controller

The central controller is responsible for running the software that monitors and coordinates the actions of the DSPR system. This includes initiating and controlling the target detection and selection functions, tuning the interferometer so that data can be collected, formatting the collected target data, and passing the data to the communications subsystem, which then sends them to the Processing and Operations Center. At the same time, the primary system central controller monitors the central controller in the other half of the DSPR and responds to operator commands. In off-line mode, it performs calibration and diagnostic functions.

In the DSPR software, there are separate procedures for each function, and the procedures operate independently. The central controller process, SYSMON, is executed as an interactive process by the operator. It, in turn, starts the other processes as detached processes. These processes operate in an asynchronous fashion. That is, activation of a procedure is event driven and does not occur in a regular sequence. For example, the interferometer data collection task becomes active only when a target is detected by the system. The VMS system of software and hardware priorities regulates interaction between various processes.

One major part of the software does not execute in the central controller, although the CPU has control over the procedure. The target detection software executes in an attached MAP 4000 array processor, which is mounted in the VAX chassis. (At the low-altitude stations, a single MAP 4000 serves each system. At high-altitude stations, in order to search both alert beams, there are two MAP 4000s on each system.) The MAP 4000 is totally software controlled by the central controller through a host interface board in the MAP.

2.2 Hardware Components of the Central Controller

Hardware components of the central controller consist of the VAX 4000 Model 200 (VAX 4200) minicomputer, an internal hard disk drive, and an internal tape drive.

2.2.1 VAX 4200 Central Controller

The VAX 4200 is designed as a general-purpose multiuser time-sharing system, although in the DSPR system it operates as a single-user system. It has a KA660 CPU module with an embedded Digital (DEC) storage systems interconnect (DSSI) adapter and an 802.3/Ethernet controller. The system has 32 MB of central memory. The CPU's processing power is 4.8 million instructions per second; it operates at a clock rate of 114 MHz. The CPU is installed in a BA430 10-slot rack-mounted chassis with a B400X 11-slot expansion chassis. The bus for peripherals is a 22-bit Qbus with a maximum input/output (I/O) bandwidth of 2.4 MB reading and 3.3 MB writing.

2.2.2 RF31 System Disk

The RF31 is a 5.25-inch half-height integrated storage element with an embedded DSSI controller. It features a 4-track, 128-kb lookahead cache, resulting in a faster time to access data, decreased seek time, and increased throughput. Its formatted capacity is 381 MB. The peak transfer rate is 4.0 MB per second and its throughput is 45 queued input/output (QIO) requests per second.

2.2.3 TK70 Tape Drive

The TK70 is a tape subsystem designed for disk backup, software distribution, and data collection. It consists of a streaming cartridge tape drive with a 5.25-inch form factor, a dual-height Qbus controller, and cables. Its maximum formatted capacity is 296 MB. Read/write speed is 100 inches per second, peak transfer rate is 125 kb/s, and recording density is 10,000 bits per inch. The TK70 uses the CompacTape II cartridge, model TK52-K. These cartridges are similar but not identical to those used in TK50 drives: the TK70 can read, but not write, TK50-formatted data.

In the DSPR system, the TK70 is used primarily for software distribution. A complete system, consisting of the VMS operating system and all the DSPR applications software, is written to tape from the engineering development model system at NRL, and is then installed at the receiver site.

3. LANGUAGES

The VMS operating system supports a number of programming languages. In the DSPR system, the programs that run in the central controller are written in VAX Fortran or VAX Macro.

3.1 VAX Fortran

VAX Fortran supports the ANSI-standard (conforming to ANSI X3.9-1978) Fortran-77, as well as providing support for many industry standard Fortran features based on Fortran-66. It includes extensions for improved performance and programmer productivity. VAX Fortran supports all VMS

Record Management Services (RMS) file formats and supports a record structure. It can access object modules generated by other languages (such as VAX Macro) and can create shareable images usable by other programs. It can also invoke all callable system routines, and can use all programming utilities.

VAX Fortran provides a number of extensions to the ANSI standard. Extensions used in the DSPR system software are listed below. Descriptions are drawn from *VAX Fortran Language Reference Manual*, which provides complete information on usage of Fortran under VMS.

- additional data types: byte and integer*2 variables and octal and hexadecimal constants
- the INCLUDE statement, which directs the compiler to stop reading statements from the current file and read the statements from the file referenced by the INCLUDE statement. When it reaches the end of that file, the compiler resumes compilation with the next statement after the INCLUDE statement.
- structure declarations, which define one or more fields within a Fortran record. This declaration defines the field names, types of data within fields, and the order and alignment of fields within a record.
- the RECORD statement, which creates a record having the form specified in a previously declared structure. Using a RECORD statement is comparable to using an ordinary Fortran type declaration, except that composite or aggregate data items are declared instead of scalar data items.
- octal notation for integer constants, of the form *'nn'O* where *nn* is a string of digits in the range 0 to 7.
- hexadecimal notation for integer constants, of the form *'nn'*X where *nn* is a string of digits plus the alphabetic characters A through F in the range 0 to F.
- the PARAMETER statement, which assigns a symbolic name to a constant. The form of the constant, rather than implicit or expiicit typing of the symbolic name, determines the data type of the variable.
- the DO WHILE statement, which executes conditionally for as long as a logical expression contained in it continues to be true.

3.2 VAX Macro

VAX Macro is an assembly language for programming the VAX computer under the VMS operating system. The instruction set includes approximately 130 instructions and 70 directives, allowing complex programming statements. Programs written in Macro can invoke any callable system routine or call any object module written in another VAX language.

The instruction set provides statements that manipulate data to perform functions such as addition, data conversion, and transfer of control. Direct assignment statements are used to equate symbols to values within the programs. General assembler directives allow the programmer to store data or reserve memory for data storage; control the alignment of parts of the program in memory; specify methods of accessing the section of memory in which the program will be stored; specify the entry point of the program or a part of the program; specify the way in which symbols are referenced, and display information messages. In addition, macro directives are available to allow the programmer to repeat identical sequences of source statements in a program without having to rewrite the sequences, and to use string operators to manipulate and test the contents of source statements.

The Macro language is described in detail in *VAX Macro and Instruction Set Reference Manual*.

4. PROCESS REQUIREMENTS

In VMS, a process is the environment where an image executes. In the DSPR system, the software executes as a number of separate processes, each of which performs a particular function of the software. (Section 1.4 lists and describes the individual processes.) At system startup, the system monitor and control process (SYSMON) is executed as an interactive process by the operator. SYSMON then starts the other processes as "detached" processes, that is, as processes that are independent of the process that created them.

Although independent, the processes must interact with each other in various ways. They must share data, which means that if one process needs to update a data file, the other processes must be kept from accessing that file while it is being updated. Events happening in different processes must be synchronized, and a precise system time must be maintained. The order in which processes execute is event-driven and does not occur in a regular sequence, therefore, the processes must be able to communicate with each other to give notification when certain events take place. Some events can take place simultaneously; in other cases, one event must be completed before another may be started.

4.1 Sharing Data

The DSPR system is designed so that various components of the software have access to one or more of five system databases. The primary database, used by all components of the software, is called DSPCOM. This database contains both static and dynamic data. Some constant parameters are established within the code; others are established at run time. Other data change constantly; for example, the contents of the storage region for array processor target lists is updated 40 times a second. Since all components of the software have access to DSPCOM, it is the primary way that critical information is made available to all components simultaneously. A detailed description of the contents and layout of the DSPCOM database is given in Appendix A of Volume 1.

Other databases are used by specific components of the software. Three target databases (TARGET1, TARGET2, and TARGET3) are used to store the data from the three digital filters assigned to each system. The individual data collection processes (ADC1, ADC2, and ADC3) are linked to their separate databases; the Data Processing (DP) procedure is linked to all three. Database NAVCOM is used to pass data that are ready for transmission to the Space Surveillance Processing Center from procedure DP to the Data Line Communications (DLC) software. For secondary system data (half- and quarter-Doppler), the procedure GETSEC collects the data from the secondary system's DP program and places them in the NAVCOM database for transmission.

The databases are set up as shared, nonexecutable program sections that contain only data. Each has a Fortran version, set up as a common block (which is translated by the compiler into a program section (PSECT)), and a matching Macro version, defined as a program section. The corresponding Fortran and Macro program sections must both have identical attributes (that is, the Macro program section must be given the same attributes as are automatically given to the Fortran common block by the compiler), and they must both contain the same variables. (It is up to the programmer to assure that corresponding Fortran and Macro program sections do indeed match!) Since they have the same name and attributes, they are gathered together by the VMS linker to form an image section; the data in the image section can then be referenced by any Fortran routine that includes the common block or by any Macro routine that includes the PSECT In the DSPR system, since there are multiple processes that must share the same data, these program sections are installed as shareable images (a shareable image is defined as a non-executable image that can be linked with executable images). The method by which they are made available simultaneously to all components of the software is by installing each common block as a shareable image and linking each program that references the common block against that shareable image.

In addition, the Target Selection (TRGSEL) procedure uses a shared memory window to pass data from the MAP 4000 array processor to the VAX 4200 during the target detection phase of the software. The shared memory window is a region of MAP memory that is addressable from both the MAP and the VAX. Either the MAP or the VAX can modify locations in the shared memory region, and both devices can read from it. This technique is used to pass parameters from the VAX to the MAP and to pass the target lists from the MAP to the VAX.

4.2 Interprocess Communication and Synchronizing Access to Shared Data

With all components of the software having simultaneous access to data in shared common blocks, communication between the processes becomes very important to properly synchronize access to the databases. For example, when one process needs to write new data to a database file, the other processes must be kept from reading or writing to that file until the data write is complete.

In the VMS operating system, a process is allowed to communicate with itself, with other processes, and with the system. Techniques for making these communications include mailboxes, local and common event flags, timers, and asynchronous system traps.

4.2.1 Mailboxes

In the VMS operating system, a mailbox is a virtual device that provides a controlled and synchronized method for processes to exchange data. Mailboxes transfer information much like hardware input/output devices, but they are a software-implemented way to perform read and write operations. Following are the basic mailbox operations (descriptions are drawn from *VMS I/O User's Reference Manual, Part 1*):

- Receive mail: One process initiates a read request to a mailbox to obtain data sent by another process. The process reads the data if a message was previously transmitted to the mailbox.

- Receive notification of mail: A process specifies that it be notified through an AST when a message is sent to its mailbox.

- Send mail, without notification: One process initiates a write request to another mailbox to send data to the second process. The sending process does not wait until the data are read by the receiving process before completing the I/O operation.

- Send mail, with notification: One process initiates a write request to another mailbox to transmit data to that process. The sending process waits until the receiving process reads the data before completing the I/O operation.

In the DSPR system, each process has a mailbox to which the other processes can send messages consisting of small packets of data and commands. An example of the use of mailboxes for interprocess communication is the start-task mechanism used by the system monitor procedure. As SYSMON starts each other process, it creates a mailbox for that process and then sends an initialization request packet to the process. When the procedure starts running, it reads its mailbox to receive the initialization packet, attempts to perform its startup functions, and sends back an initialization response indicating success or failure in starting. SYSMON can then send a failure message to the operator for any task that has not started correctly.

VMS system service routines are used to create and delete mailboxes. These are described in Section 6.1, Input/Output Services.

4.2.2 Event Flags

Processes are also linked through event flags. VMS offers two types of event flags: local event flags, which are specific to a single process, and common event flags, which are used to synchronize events between separate processes. Each event flag has its unique number and is associated with a cluster of 32 event flags. Event flags 0 through 31 and 32 through 63 are local event flags; flags 64 through 95 and 96 through 127 are common event flags. Before a common event flag can be used, a logical name must be associated with the common event flag cluster. The first program to name a common event flag cluster creates it, thus clearing all the flags in the cluster. Then other processes can reference the cluster and use the event flags in the cluster.

The DSPR software uses a number of the common event flags. Those used are listed, with their purpose, mnemonic names, and event flag numbers in Appendix G, Interprocess Communications, of Volume 1 (NRL/FR/8154--93-9577). Since each event flag can have only two values—set (on) or cleared (off)—the amount of information that can be transferred is limited. Therefore they are used only for transmitting information that has on/off values or for signaling the occurrence of an event. After the occurrence of an event, appropriate variables may be checked to determine the action to be taken. For example, event flag DCOLEF is set by an ADC task to indicate that data collection has completed and data are available. Process DP waits for this flag to be set before it starts processing data.

The system service routines used to manipulate event flags are described in Section 6.2.

4.2.3 Timers

The VMS operating system allows processes to schedule events for a specific time of day or after a specified time interval through set-timer and cancel-timer system service routines. Timer services require that the time be specified in a 64-bit format; therefore, various conversion routines are available.

The DSPR software uses timers, along with event flags and asynchronous traps, to make sure that events take place in the proper sequence. Timers are also used to insert wait time into a process to allow for the time it takes for data to be passed.

4.2.4 Asynchronous System Traps (ASTs)

Asynchronous system traps are software interrupts that occur asynchronously to the execution of a program. They are used to signal a program to execute a routine whenever a certain condition occurs. An "AST routine" is one that is not called explicitly by the current program, but instead is called by the operating system after an AST is received by the program. When the AST routine has completed, the program that was interrupted resumes its execution from the point at which it was interrupted.

AST routines differ from regular subroutines in that they are not called directly. Also, because they cannot return a function value or pass an argument, the DSPR global databases (shareable images) are used to pass arguments between the AST routine and the routine that causes it to be executed.

In the DSPR software, an AST routine is used for handling unsolicited keyboard input from the system operator. For example, in the system monitor routine DISPLAY, which displays the status of the DSPR system on the primary system's CRT, if the operator enters any keyboard character, the CRT is cleared and control returns to the dispatch routine, which allows the operator to select a function. When DISPLAY starts, it calls routine UNSOL, which includes an AST routine called ASTCHAR. When a character is typed, the AST routine clears the character just typed, checks to see if the display is running, and, if it is, sets a variable DSTAT to indicate that the display should be cleared. Control is returned to the display routine, DSTAT is checked, and control is then transferred to the dispatch routine.

5. INPUT/OUTPUT PROCESSING AND DEVICE DRIVERS

The VMS operating system supports several types of input/output requests. I/O operations may be handled indirectly through VMS RMS, which provides a set of macros for general purpose, device-independent functions. I/O may be performed directly through system service routines that execute such functions as assigning and deassigning channels, queuing I/O requests, and allocating or deallocating devices. VAX Fortran programming language statements such as READ and WRITE are implemented using the VMS RMS facilities. Fortran programmers can also access the system service routines directly. In the DSPR system, because of speed requirements, most I/O is done with system service routines. Section 6.1 briefly describes those used in the DSPR software.

The DSPR software uses I/O requests to communicate with the various devices attached to and controlled by the CPU. The devices include the operator's terminal and hardcopy printer, the signal calibrator, custom cards on the utility bus, the modem that transfers data to the Processing and Operations Center at Dahlgren, and the Ethernet that allows communications between the primary and secondary system CPUs.

Input/output to a peripheral device is accomplished through a device driver—a set of tables and routines that performs the functions of defining the peripheral device to the operating system, initializing the device or its controller at system startup, translating software requests for I/O operations into device-specific commands, activating the device, responding to hardware interrupts generated by the device, reporting device errors, and returning data and status information from the device to the user program.

As an example of an I/O request using a device driver, following are the steps for outputting a text string to the operator terminal. Before the I/O request can be made to the terminal, a channel must be assigned to establish a link between the process and the terminal; the channel is then used to transfer whatever information is specified to the terminal. The $ASSIGN system service routine is used to assign a channel number to the terminal. Then the process calls the $QIOW system service routine, which queues an I/O request that waits for the I/O operation to actually complete before returning to the calling routine. The $QIOW has various arguments that specify the channel number that was assigned; the I/O function, in this case a write; an event flag number that can be tested to see if the function has completed; and the address of an information buffer that contains carriage control information, the line and column numbers at which to place the text, and the text string to be displayed on the terminal.

The individual system service routines used for direct I/O or with device drivers are detailed in the *VMS System Services Reference Manual*. Device drivers in general are discussed in the *VMS I/O User's Reference Manual, Parts 1 and 2*, the *VMS Device Support Reference Manual*, and the *VMS Device Support Manual*.

The following device drivers ~ e used in the DSPR software:

- Terminal driver, described in *VMS I/O User's Reference Manual, Parts 1 and 2*. This driver is used for the operator's CRT terminal and the hardcopy terminal.

- DSV11 driver, described in *VAX Wide Area Network Device Drivers*. The DSV11 controls communications with the modem.

- IEX driver, described in *IEX/VMS User's and Installation Guide*. This driver allows Fortran and VAX Macro programs to communicate through an IEQ11-A with devices that contain an IEEE-488 interface. In the DSPR system, these are the signal calibrator, the time-of-day clock, and the frequency standard.

- DRV11-WA driver, described in *VMS I/O User's Reference Manual, Part 2*. The DSPR system contains four DRV1W interfaces, three for the digital filters and one for the utility bus.

6. SYSTEM SERVICES

System services are procedures used by the VMS operating system. When called in application programs, they allow the programmer to control resources available to processes, to provide for communication among processes, and to perform basic operating system functions.

In the DSPR system, five types of system services are used. These are the procedures that provide for communication between the various DSPR processes, as well as synchronization of events between processes. The individual services used in the software are listed in the sections below; the descriptions are drawn from *VMS System Services Reference Manual*. These routines can be called from either Fortran or VAX Macro programs; the format shown here is for Fortran calls. General information about system services may be found in *Introduction to VMS System Services*.

6.1 Input/Output Services

Allocate Device ($ALLOC): This service allocates a device for exclusive use by a process and its subprocesses. No other process can allocate the device or assign channels to it until the image that called $ALLOC exits or explicitly deallocates the device with the deallocate device ($DALLOC) service.

```
call sys$alloc  ( %descr(devnam), [phylen], %descr([phybuf]), %val([acmode]),
%val([flags]) )
```

Assign I/O Channel ($ASSIGN): This service either provides a process with an I/O channel so that input/output operations can be performed on a device, or it establishes a logical link with a remote node on a network.

```
call sys$assign  ( %descr(devnam), chan, %val([acmode]), %descr([mbxnam]) )
```

Cancel I/O On Channel ($CANCEL): This service cancels all pending I/O requests on a specified channel. In general, this includes all I/O requests that are queued as well as the request currently in progress. To cancel I/O on a channel, the access mode of the calling process must be equal to or more privileged than the access mode of the process that made the original channel assignment.

```
call sys$cancel ( %val(chan) )
```

Create Mailbox and Assign Channel ($CREMBX): This service creates a virtual mailbox device named MBA*n* and assigns an I/O channel to it. The system provides the unit number *n* when it creates the mailbox. If a mailbox with specified name already exists, the $CREMBX service assigns a channel to the existing mailbox.

```
call sys$crembx ( %val([prmflg]), chan, %val([maxmsg]), %val([bufquo]),
%val([promsk]), %val([acmode]), %descr([lognam]) )
```

Create and Map Section ($CRMPSC): This service allows a process to map a section of its address space with a specified section of a file or specified physical addresses represented by page frame numbers. This service also allows the process to create either type of section and to specify that the section be available only to the creating process or to all processes that map to the global section.

```
call sys$crmpsc ( [inadr], [retadr], %val([acmode]), %val([flags]), %descr([gsdnam]),
[ident], %val([relpag]), %val([chan]), %val([pagcnt]), %val([vbn]), %val([prot]),
%val([pfc]) )
```

Deallocate Device ($DALLOC): This service deallocates a previously allocated device. The issuing process relinquishes exclusive use of the device, thus allowing other processes to assign or allocate that device.

```
call sys$dalloc ( %descr([drvnam]), %val([acmode]) )
```

Delete Mailbox ($DELMBX): This service marks a permanent mailbox for deletion. The actual deletion of the mailbox and of its associated logical name assignment occurs when no more I/O channels are assigned to the mailbox.

```
call sys$delmbx ( %val(chan) )
```

Formatted ASCII Output Services ($FAO): This service converts a binary value into an ASCII character string.

```
call sys$fao ( %descr(ctrstr), [outlen], outbuf, %val([p1])...[pn]) )
```

Queue I/O Request ($QIO): This service queues an I/O request to a channel associated with a device. The $QIO service completes asynchronously, that is, it returns to the caller immediately after queuing the I/O request without waiting for the I/O operation to complete.

```
call sys$qio ( %val([efn]), %val(chan), %val(func), [iosb], [astadr], %val[(astprm)],
[p1], [p2], [p3], [p4], [p5], [p6] )
```

Queue I/O Request and Wait ($QIOW): This service queues an I/O request to a channel associated with a device. The $QIOW service completes synchronously, that is, it returns to the caller after the I/O operation has actually completed.

```
call sys$qiow (%val([efn]), %val(chan), %val(func), [iosb], [astadr], %val([astprm]),
[p1], [p2], [p3], [p4], [p5], [p6] )
```

6.2 Event Flag Services

Associate Common Event Flag Cluster ($ASCEFC): This service causes a named common event flag cluster to be associated with a process for the execution of the current image and to be assigned a process-local cluster number for use with other event flag services. If the named cluster does not exist but the process has suitable privilege, the service creates the cluster.

```
call sys$ascefc ( %val(efn), %descr(name), %val([prot]), %val([perm]) )
```

Clear Event Flag ($CLREF): This service clears (sets to 0) an event flag in a local or common event flag cluster.

```
call sys$clref ( %val(efn) )
```

Read Event Flags ($READEF): This service returns the current status of all 32 event flags in a local or common event flag cluster. In addition, the condition value returned indicates whether the specified event flag is set or clear.

```
call sys$readef ( %val(efn), state )
```

Set Event Flag ($SETEF): This service sets an event flag in a local or common event flag cluster. The condition value returned by $SETEF indicates whether the specified flag was previously set or clear. After the event flag is set, processes waiting for the event flag to be set resume execution.

```
call sys$setef ( %val(efn) )
```

Wait for Single Event Flag ($WAITFR): This service tests a specific event flag and returns immediately if the flag is set. Otherwise, the process is placed in a wait state until the event flag is set.

```
call sys$waitfr ( %val(efn) )
```

Wait for Logical AND of Event Flags ($WFLAND): This service allows a process to specify a set of event flags for which it wants to wait. The process is put in a wait state until all specified event flags are set, at which time $WFLAND returns to the caller and execution resumes.

```
call sys$wfland ( %val(efn), %val(mask) )
```

Wait for Logical OR of Event Flag ($WFLOR): This service allows a process to specify a set of event flags for which it wants to wait. The process is put in a wait state until any one of the specified event flags is set, at which time $WFLOR returns to the caller and execution resumes.

```
call sys$wflor ( %val(efn), %val(mask) )
```

6.3 Timer and Time Conversion Services

Convert Binary Time to ASCII String ($ASCTIM): This service converts an absolute or delta time from 64-bit system time format to an ASCII string.

```
call sys$asctim ( [timlen], %descr(timbuf), [timadr], %val([cvtflg]) )
```

Convert ASCII String to Binary Time ($BINTIM): This service converts an ASCII string to an absolute or delta time value in the system 64-bit time format suitable for input to the set timer ($SETIMR) or schedule wakeup ($SCHDWK) service.

```
call sys$bintim ( %descr(timbuf), timadr )
```

Cancel Timer ($CANTIM): This service cancels all or a selected subset of the set timer requests previously issued by the current image executing in a process. Cancellation is based on the request identification specified in the set timer ($SETIMR) service. If the programmer gives the same request identification to more than one timer request, all requests with that request identification are cancelled. The calling process can cancel only timer requests that are issued by a process whose access mode is equal to or less privileged than that of the calling process.

```
call sys$cantim ( %val([reqidt]), %val([acmode]) )
```

Get Time ($GETTIM): This service returns the current system time in a VMS-specific 64-bit format.

```
call sys$gettim ( timadr )
```

Convert Binary Time to Numeric Time ($NUMTIM): This service converts an absolute or delta time from a VMS-specific 64-bit system time format to binary integer date and time values.

```
call sys$numtim ( timbuf, [timadr] )
```

Set Timer ($SETIMR): This service sets the timer to expire at a specified time. When the timer expires, an event flag is set and (optionally) an AST routine executes.

```
call sys$setimr ( %val([efn]), daytim, [astadr], %val([reqidt]), %val([flags]) )
```

6.4 Process Control Services

Create Process ($CREPRC): This service creates a subprocess or detached process on behalf of the calling process.

```
call sys$creprc ( [pidadr], %descr([image]), %descr([input]), %descr([output]),
%descr([error]), [prvadr], [quota], %descr([prcnam]), %val([baspri), %val([uic]),
%val([mbxunt]), %val([stsflg]))
```

Delete Process ($DELPRC): This service allows a process to delete itself or another process.

```
call sys$delprc ( [pidadr], %descr([prcnam]) )
```

Set Priority ($SETPRI): This service changes the base priority of the process. The base priority is used to determine the order in which executable processes are to run.

```
call sys$setpri ( [pidadr], %descr([prcnam]), %val(pri), [prvpri] )
```

Set Privileges ($SETPRV): This service enables or disables specified privileges for the calling process.

```
call sys$setprv ( %val([enbflg]), [prvadr], %val([prmflg]), [prvprv] )
```

Set Process Swap Mode ($SETSWM): This service allows a process to control whether it can be swapped out of the balance set.

```
call sys$setswm ( %val([swpflg]) )
```

6.5 Asynchronous System Trap Services

Set AST Enable ($SETAST): This service enables or disables the delivery of ASTs for the access mode from which the service call was issued.

```
call sys$setast ( %val(enbflg) )
```

REFERENCES

Manuals Published by Digital Equipment Corporation, Maynard, Massachusetts:

IEX/VMS User's and Installation Guide, DEC document AA-X673D-TE, July 1990.

Introduction to VMS System Services, DEC document AA-LA68A-TE, April 1988.

VAX Fortran Language Reference Manual, DEC document AA-D034E-TE, June 1988.

VAX Macro and Instruction Set Reference Manual, DEC document AA-LA89A-TE, April 1988.

VAX Wide Area Network Device Drivers, DEC document AA-LN26B-TE, January 1989.

VMS Device Support Reference Manual, DEC document AA-PBPXA-TE, June 1990.

VMS Device Support Manual, DEC document AA-PWBWA-TE, June 1990.

VMS I/O User's Reference Manual, Part 1, DEC document AA-LA84B-TE, June 1990.

VMS I/O User's Reference Manual, Part 2, DEC document AA-LA85A-TE, April 1988.

VMS System Services Reference Manual, DEC document AA-LA69A-TE, April 1988.

Other Manuals:

MAP-4000 Software Utilities, CSP Inc., Billerica, Massachusetts, 1990.

ACRONYMS AND ABBREVIATIONS

AST	asynchronous trap
CPU	central processing unit
DCL	Digital (DEC) command language
DEC	Digital Equipment Corporation
DLC	data line communications
DP	data processing
DSPR	Digital Signal Processing Receiver
DSSI	Digital (DEC) storage systems interconnect
I/O	input/output
kb	kilobytes
kb/s	kilobytes per second
MB	megabytes
MHz	megahertz
NAVSPACECOM	Naval Space Command (formerly Naval Space Surveillance (NAVSPASUR))
NRL	Naval Research Laboratory
pps	pulse per second
PSECT	program section
QIO	queued input/output
RMS	record management services
SPADATS	Space Detection and Tracking System
SYSMON	system monitor and control
TRGSEL	target selection
VAX	Virtual Address eXtension
VMS	virtual memory system